

KOMPILÁTORY: Syntaktická analýza (Metódy zdola-nahor 2)

Jana Dvořáková

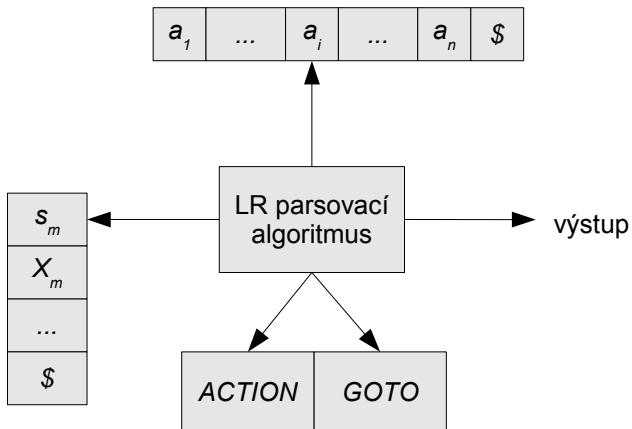
dvorakova@dcs.fmph.uniba.sk



LR parsovanie

- Tri techniky konštrukcie LR parsera
 - 1 Jednoduchý LR parser (simple LR - SLR)
 - Najjednoduchší na implementáciu, ale najslabší
 - 2 Kanonický LR parser (LR)
 - Najsilnejší, ale najnáročnejší na implementáciu (veľká parsovacia tabuľka)
 - 3 LR parser s výhľadom (lookahead LR - LALR)
 - Kompromis medzi SLR a LR

Model LR parsera



SLR (simple LR) parovanie

- Najslabšia metóda, ale najľahšie implementovateľná
- Konštruujeme *SLR parovaciu tabuľku*
- Dá sa použiť pre *SLR gramatiky*
 - Ak sa pri zostrojovaní parovacej tabuľky objavia konflikty, vstupná gramatika nie je SLR
- Základnou ideou je skonštruovať pre vstupnú gramatiku DKA rozpoznávajúci životaschopné prefixy
- LR a LALR metódy sú oproti SLR rozšírené o výhľad dopredu (lookahead)
 - Myslí sa výhľad v stavoch zostrojovaného DKA, nie výhľad na vstupné symboly !

LR(0) položka

- Pravidlo s bodkou na pravej strane

Pr.: $A \rightarrow XYZ$ $[A \rightarrow .XYZ]$
 $[A \rightarrow X.YZ]$
 $[A \rightarrow XY.Z]$
 $[A \rightarrow XYZ.]$

- Reprezentované dvojicou: (*číslo pravidla, číslo pozície bodky*)
- Neformálne: Aká veľká časť z pravej strany pravidla je už známa

- Úplná položka

- Položka s bodkou na konci pravej strany pravidla

- Platná položka (pre daný životaschopný prefix)

- Položka $[A \rightarrow \beta_1.\beta_2]$ je platná pre životaschopný prefix $\alpha\beta_1$ ak existuje odvodenie

$$S' \Rightarrow_{rm}^* \alpha A w \Rightarrow_{rm}^* \alpha \beta_1 \beta_2 w$$

- Vo všeobecnosti je položka platná pre viacero životaschopných prefixov

Konštrukcia konečného automatu

- Najskôr vstupnú gramatiku G rozšíríme o nový počiatkový neterminál S' a pravidlo $S' \rightarrow S$ a označíme G'
- Položky = stavy NKA rozpoznávajúceho životaschopné prefixy
 - Počiatčným stavom je položka $[S' \rightarrow .S]$
 - Koncovými stavmi sú úplné položky



Ale my potrebujeme DKA!

- Množiny položiek = stavy hľadaného DKA
- Konštruujeme *kanonické množiny položiek* pre rozšírenú gramatiku G' , pričom definujeme
 - 1 Pomocnú operáciu *closure*
 - 2 Operáciu *goto* na konštrukciu prechodovej funkcie DKA
 - 3 Operáciu *items* na konštrukciu stavov DKA
- Ekvivalent k štandardnej konštrukcii DKA k NKA

Operácia *closure*

```
function closure(I);  
begin  
     $J := I$ ;  
    repeat  
        for každú položku  $[A \rightarrow \alpha.B\beta]$  v  $J$  a každé pravidlo  
             $B \rightarrow \gamma$  gramatiky  $G$  také, že  $[B \rightarrow .\gamma]$  nie je v  $J$  do  
                pridaj  $[B \rightarrow .\gamma]$  do  $J$   
    until do  $J$  nie je možné pridať žiadnu ďalšiu položku;  
    return  $J$   
end
```

Operácia *closure*

Implementácia

- Zvyčajne sa implementuje pomocou bool. poľa *added*, indexovaného neterminálmi:
 - $added[B] = true$ ak pridáme položky $[B \rightarrow \cdot \gamma]$ pre každé B -pravidlo $B \rightarrow \gamma$
 - Postačuje pamätať si zoznam takýchto neterminálov
- Rozlišujeme dva druhy položiek
 - 1 Jadrové položky:
 - Počiatočná položka $[S' \rightarrow \cdot S]$ a všetky položky, ktoré nemajú bodku na začiatku pravej strany.
 - 2 Mimojadrové položky:
 - Položky, ktoré majú bodku na začiatku pravej strany (okrem poč. položky).
- Uzáver (closure) môže byť reprezentovaný iba jadrovými položkami
 - Nejadrové sa vždy dajú jednoducho odvodiť

Operácia *goto*

$$\textit{goto}(I, X) = \textit{closure}(\{[A \rightarrow \alpha X.\beta] \mid [A \rightarrow \alpha.X\beta] \in I\})$$

- Argumentami sú množina položiek I a symbol gramatiky X
- Vrátí uzáver množiny položiek, do ktorých sa vieme dostať z položiek v A prechodom cez X
- Alebo aj:
 - Ak I je množina položiek platných pre nejaký životaschopný prefix γ , potom $\textit{goto}(I, X)$ je množina položiek platná pre životaschopný prefix γX .
- Reprezentuje prechodovú funkciu konštruovaného DKA

Operácia *items*

```
procedure items( $G'$ );  
begin  
   $C := \{closure(\{[S' \rightarrow .S]\})\};$   
  repeat  
    for každú množinu položiek  $I$  v  $C$  a každý symbol  
      gramatiky  $X$  také, že množina  $goto(I, X)$  nie je prázdna  
      a nie je v  $C$  do  
        pridaj  $goto(I, X)$  do  $C$   
  until do  $C$  nie je možné pridať žiadnu ďalšiu množinu položiek  
end
```

Platné položky

- Význam tvrdenia " $A \rightarrow \beta_1.\beta_2$ je platná pre životaschopný prefix $\alpha\beta_1$ ":
 - Rozhodovanie medzi akciami shift/reduce
 - $\beta_2 \neq \varepsilon$: shift
 - $\beta_2 = \varepsilon$: reduce podľa $A \rightarrow \beta_1$
- Kde môžu vzniknúť konflikty:
 - Shift/reduce: máme dve rôzne platné položky a každá určí inú akciu
 - Reduce/reduce: máme viac úplných platných položiek
 - Niektoré môžu byť vyriešené rozšírením o výhľad v stavoch DKA (LR, LALR)
- Množina platných položiek pre životaschopný prefix γ je rovná množine položiek - stavu, ktorý DKA dosiahne prechodom z počiatočného stavu na jednotlivé symboly γ .

Konštrukcia SLR parsovacej tabuľky

Vstup: Rozšírená gramatika G' .

Výstup: SLR parsovacia tabuľka pre G' reprezentovaná funkciami *action* a *goto*.

- 1 Skonštruuj kanonické množiny LR(0) položiek pre G' : $C = \{I_0, I_1, \dots, I_n\}$.
- 2 Nech stav i zodpovedá množine položiek I_i . Akcia pre stav i sa určí nasledovne:
 - a) Ak $[A \rightarrow \alpha.a\beta] \in I_i$ a $goto(I_i, a) = I_j$, potom $action[i, a] = \text{shift } j$. Symbol a musí byť terminál.
 - b) Ak $[A \rightarrow \alpha.] \in I_i$, potom $action[i, a] = \text{reduce } A \rightarrow \alpha$ pre všetky $a \in FOLLOW(A)$, $A \neq S'$.
 - c) Ak $[S' \rightarrow .S] \in I_i$, potom $action[i, \$] = \text{accept}$.

Ak uvedené pravidlá spôsobia nejaké konflikty, gramatika nie je SLR(1) a parser nie je možné zostrojiť.

- 3 Goto prechody pre stav i sú zostrojené pre všetky neterminály A nasledovne: Ak $goto(I_i, A) = I_j$, potom $goto[i, A] = j$.
- 4 Všetky záznamy nedefinované v krokoch 2 a 3 predstavujú chybu - akcia error.
- 5 Počiatočný stav parsera je množina položiek obsahujúca $[S' \rightarrow .S]$.

Gramatiky mimo triedy SLR

- Mimo triedy SLR sú všetky nejednoznačné ale aj niektoré jednoznačné gramatiky

Príklad jednoznačnej, ale nie SLR gramatiky a jej kanonické množiny položiek:

$S \rightarrow L = R$

$S \rightarrow R$

$L \rightarrow *R$

$L \rightarrow id$

$R \rightarrow L$

$I_0 : S' \rightarrow .S$

+ mimojadrové položky pre S, L, R

$I_1 : S' \rightarrow S.$

$I_2 : S \rightarrow L. = R$
 $R \rightarrow L.$

$I_3 : S \rightarrow R.$

$I_4 : L \rightarrow *.R$
+ mimojadrové položky pre R, L

$I_5 : L \rightarrow id.$

$I_6 : S \rightarrow L = .R$
+ mimojadrové p.
pre R, L

$I_7 : L \rightarrow *R.$

$I_8 : R \rightarrow L.$

$I_9 : S \rightarrow L = R.$

- Pri I_2 vznikne shift/reduce konflikt
 - $action[2, =] = \text{shift } 6$, podľa prvej položky
 - $action[2, =] = \text{reduce } R \rightarrow L$, podľa druhej položky, '=' $\in FOLLOW(R)$
- Pri LR a LALR parsovaní konflikt nevznikne

LR parovanie

- Najuniverzálnejšia metóda
- Problém pri SLR parovaní:
 - Na vrchu zásobníka máme životaschopný prefix $\beta\alpha$
 - Sme v stave i , pričom $[A \rightarrow \alpha.] \in I_i$
 - Čítaný vstupný symbol $a \in FOLLOW(A)$
Ale čo ak βA nie je v skutočnosti v žiadnej pravej vetnej forme nasledované a !
- Pri LR parovaní je v stave je uložená ďalšia informácia - *výhľad položky*
 - Terminál, ktorý môže nasledovať za pravou stranou pravidla
- Dostávame LR(1) položky
- Každá SLR gramatika je aj LR gramatikou, ale pre danú gramatiku LR parser môže mať viac stavov ako SLR parser

LR(1) položka

- Dvojica (*LR(0) položka, terminál*)
 - $[A \rightarrow \alpha.\beta, a]$, kde $A \rightarrow \alpha\beta$ je pravidlo a a je terminál
 - LR(1) - položka má výhľad 1
- Výhľad položky sa využíva pri úplných LR(1) položkách (pri redukcii)
 - Pri položke $[A \rightarrow \alpha., a]$ sa redukuje podľa pravidla $A \rightarrow \alpha$ iba v prípade, ak práve čítaný vstupný symbol je a
- *Platná* LR(1) položka
 - LR(1) položka $[A \rightarrow \alpha.\beta, a]$ je platná pre životaschopný prefix γ ak existuje odvedenie $S \Rightarrow_{rm}^* \delta A w \Rightarrow_{rm}^* \delta \alpha \beta w$, kde
 - 1 $\gamma = \delta \alpha a$
 - 2 buď a je prvý symbol w , alebo $w = \varepsilon$ a $a = \$$.

Operácie *closure*, *goto*

```
function closure(I);  
begin  
    repeat  
        for každú položku  $[A \rightarrow \alpha.B\beta, a] \in I$ , každé pravidlo  $B \rightarrow \gamma$   
            gramatiky  $G'$  a každý terminál  $b \in FIRST(\beta a)$  také, že  
             $[B \rightarrow \cdot\gamma, b]$  nie je v  $I$  do  
                pridaj  $[B \rightarrow \cdot\gamma, b]$  do  $I$   
    until do  $I$  nie je možné pridať žiadnu ďalšiu položku;  
    return  $I$   
end  
  
function goto( $I, X$ );  
begin  
    nech  $J$  je množina položiek  $[A \rightarrow \alpha X.\beta, a]$  taká, že  
     $[A \rightarrow \alpha.X\beta, a] \in I$   
    return closure( $J$ )  
end
```


Operácia *items*

```
procedure items( $G'$ );  
begin  
   $C := \{closure(\{[S' \rightarrow .S, \$]\})\};$   
  repeat  
    for každú množinu položiek  $I$  v  $C$  a každý symbol  
      gramatiky  $X$  také, že množina  $goto(I, X)$  nie je prázdna  
      a nie je v  $C$  do  
        pridaj  $goto(I, X)$  do  $C$   
  until do  $C$  nie je možné pridať žiadnu ďalšiu množinu položiek  
end
```

Konštrukcia LR parsovacej tabuľky

Vstup: Rozšírená gramatika G' .

Výstup: Kanonická LR parsovacia tabuľka pre G' reprezentovaná funkciami *action* a *goto*.

- 1 Skonštruuj kanonické množiny LR(1) položiek pre G' : $C = \{I_0, I_1, \dots, I_n\}$.
- 2 Nech stav i zodpovedá množine položiek I_i . Akcia pre stav i sa určí nasledovne:
 - a) Ak $[A \rightarrow \alpha.a\beta, b] \in I_i$ a $goto(I_i, a) = I_j$, potom $action[i, a] = \text{shift } j$. Symbol a musí byť terminál.
 - b) Ak $[A \rightarrow \alpha., a] \in I_i$, potom $action[i, a] = \text{reduce } A \rightarrow \alpha$, $A \neq S'$.
 - c) Ak $[S' \rightarrow S., \$] \in I_i$, potom $action[i, \$] = \text{accept}$.

Ak uvedené pravidlá spôsobia nejaké konflikty, gramatika nie je LR(1) a parser nie je možné zostrojiť.

- 3 Goto prechody pre stav i sú zostrojené pre všetky neterminály A nasledovne: Ak $goto(I_i, A) = I_j$, potom $goto[i, A] = j$.
- 4 Všetky záznamy nedefinované v krokoch 2 a 3 predstavujú chybu - akcia error.
- 5 Počiatočný stav parsera je množina položiek obsahujúca $[S' \rightarrow .S, \$]$.

LALR (lookahead LR) parsovanie

- Metóda často používaná v praxi - vhodný kompromis medzi SLR a LR
 - Parsovacia tabuľka je výrazne menšia ako pri LR parsovaní
Počet stavov LALR parsera = počet stavov SLR parsera
 - Je možné rozpoznať všetky bežné koštrukcie programovacích jazykov
- Vychádza sa z množín kanonických položiek (stavov DKA) zostrojených LR metódou a tie sa podľa určitých pravidiel zlučujú dokopy

Zlučovanie množín položiek

- Zlúčia sa stavy, ktoré majú rovnaký prvý komponent
Pr. $\{[A \rightarrow \alpha.\beta, a]\}, \{[A \rightarrow \alpha.\beta, b]\} \Rightarrow \{[A \rightarrow \alpha.\beta, a/b]\}$
- Funkcia *goto* sa upraví jednoducho
 - *goto(I, X)* závisí iba na prvých komponentoch *I*, preto ak *I* vznikla zlúčením stavov, *goto(I, X)* vznikne zlúčením pôvodných *goto* funkcií
- Prečo je LALR parser slabší:
 - Pri zlučovaní stavov môžu vzniknúť nové reduce/reduce konflikty (ale nikdy nie nové shift/reduce konflikty!)
- Porovnanie práce parserov:
 - 1 Korektný vstup
 - LR aj LALR parser správajú rovnako, vykonajú rovnakú postupnosť shift a reduce akcií
 - 2 Nekorektný vstup
 - Keď LR parser vyhlási chybu, LALR parser môže vykonať ešte niekoľko redukcií, kým tiež vyhlási chybu (ale nikdy nie shift!)

Konštrukcia LALR parsovacej tabuľky

Vstup: Rozšírená gramatika G' .

Výstup: LALR parsovacia tabuľka pre G' reprezentovaná funkciami *action* a *goto*.

- 1 Skonštruuj kanonické množiny LR(1) položiek pre G' : $C = \{I_0, I_1, \dots, I_n\}$.
 - 2 Nájdi stavy so spoločným prvým komponentom a tieto stavy nahrad' ich zjednotením.
 - 3 Nech $C' = \{J_0, J_1, \dots, J_m\}$ je výsledná množina LR(1) položiek. Funkcia *action* sa zostrojuje rovnako ako pri LR parseri. Ak vzniknú nejaké konflikty, gramatika nie je LALR(1) a parser nie je možné zostrojiť.
 - 4 Goto prechody sú zostrojené nasledovne: Ak J je zjednotenie niekoľkých LR(1) položiek, $J = I_1 \cup I_2 \cup \dots \cup I_k$, potom prvé komponenty $goto(I_1, X)$, $goto(I_2, X)$, \dots , $goto(I_k, X)$ sú rovnaké keďže I_1, I_2, \dots, I_k majú rovnaký prvý komponent. Nech K je zjednotenie všetkých množín s rovnakým prvým komponentom ako má $goto(I_1, X)$. Potom $goto(J, X) = K$.
- Jednoduchý, ale priestorovo neefektívny algoritmus
 - Existuje aj efektívna metóda bez priameho zostrojovania všetkých LR(1) položiek

Optimalizácia LR parsovej tabuľky

- Dvojrozmerné pole je priestorovo neefektívne
- Tabuľka pre *action*: veľa riadkov je rovnakých
 - 1 Smerníky na jednorozmerné pole
 - Rovnaké riadky sa reprezentujú ako jednorozmerné polia a v tabuľke *action* sú na ne smerníky
 - Terminály sú očíslované $0, \dots, |T| - 1$ a akcia pre daný terminál sa hľadá podľa jeho čísla (= offset v poli)
 - 2 Zoznam akcií pre každý stav resp. množinu stavov

Pr.: stavy 0,4,6,7 *symbol* *action*

	id	s5
	(s4
	any	<i>error</i>

- Tabuľka pre *goto*: veľa záznamov je prázdnych (nie sú potrebné, nikdy sa podľa nich nedetekuje chyba)
 - Zoznam posunov pre každý neterminál
- A: (súčasný stav, nasledujúci stav)

Čo s nejednoznačnými gramatikami

- Nejednoznačné gramatiky nie sú LR
- Niekedy je vhodnejšie ich použiť spolu s explicitnými pravidlami pre riešenie konfliktov, napr.:
 - Výrazy
 - Explicitne určíme prioritu a asociativitu operátorov
 - if..then..else
 - Explicitne uprednostníme shift, ak máme v zásobníku "if *expr* then *stmt*" a na vstupe "else"
- Získame efektívnejší parser a prehľadnejšiu gramatiku
- Niektoré záznamy parsovacej tabuľky je nutné doplniť "ručne"

Spracovanie chýb pri LR parsovaní (1)

- Chyba je detekovaná pri prázdnych záznamoch v *action* tabuľke (nikdy nie v *goto*!)
- Chyby sú ohlásené hneď, ako je to možné (na rozdiel od oper. precedenčného parsera)
 - LR parser nevykoná už žiadnu akciu pred ohlásením chyby
 - SLR a LALR parsery môžu vykonať ešte niekoľko redukcií

1. Zotavenie v móde paniky

- Vyberáme symboly zo zásobníka, kým nenarazíme na stav s , ktorý má *goto* prechod na určitý neterminál A
- Posúvame sa na vstupe, kým nenájdeme taký terminál a , ktorý môže nasledovať za A
- Pridáme na vrch zásobníka A a stav *goto* $[s, A]$
- Obnovíme parsovanie
- Neterminály A = synchronizujúce symboly, zväčša hlavné časti programu výraz, príkaz, blok

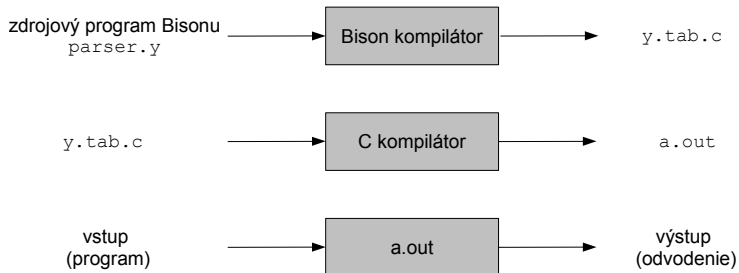
Spracovanie chýb pri LR parsovaní (2)

2. Zotavenie na úrovni frázy

- Ku každému prázdnému záznamu v parsovacej tabuľke sa pridá funkcia na ošetrovanie konkrétnej chyby
- Modifikácia symbolov na vrchu zásobníka a/alebo vstupných symbolov
- Na rozdiel od OP parsovania sa nemusíme zaoberať chybnými pokusmi o redukciu
- Treba sa vyhnúť zacykleniu

Generátory parserov

Bison



Syntaktická analýza

Zhrnutie

1 Univerzálne metódy

- CYK, Earley

2 Metódy zhora-nadol

- Všeobecná metóda: rekurzívny zostup
- Deterministické algoritmy
 - Prediktívne parsovanie - **LL(k) gramatiky**

3 Metódy zdola-nahor

- Všeobecná metóda: shift-reduce parsovanie
 - Deterministické algoritmy
 - Operátorovo precedenčné parsovanie - **OP gramatiky**
 - SLR parsovanie - **SLR(k) gramatiky**
 - LR parsovanie - **LR(k) gramatiky**
 - LALR parsovanie **LALR(k) gramatiky**
-
- V kompilátoroch sa využívajú najmä gramatiky s **k=1**

Vzťahy medzi triedami gramatík

